A RISC-V Extension to Minimize Privileges of Enclave Runtimes

Neelu S. Kalani
EPFL, Switzerland

Edouard Bugnion
EPFL, Switzerland

Abstract

In confidential computing, the view of the system software is Manichean: the host operating system is untrusted and the TEE runtime system is fully trusted. However, the runtime system is often as complex as a full operating system, and thus is not free from bugs and exploitable vulnerabilities. Yet, it executes with complete system-level control over the enclave application, in violation of the least privilege principle. While the confidential computing research community has been striving to secure trusted software from its untrusted counterpart, efforts fall short when it comes to securing the enclave application from potentially bug-prone and vulnerable trusted runtime systems.

This project describes the design of a simple RISC-V extension that prevents trusted runtime systems from accessing the enclave application's memory. We implement the hardware extension in the QEMU functional simulator and extend the Keystone TEE framework and its runtime system, Eyrie, to enforce the least privilege principle, support unmodified enclave applications, and prevent a class of Iago attacks that leverage the runtime system's unrestricted access to the enclave application's memory.

ACM Reference Format:
Neelu S. Kalani and Edouard Bugnion. 2023. A RISC-V Extension to Minimize Privileges of Enclave Runtimes. In *6th Workshop on System Software for Trusted Execution (SysTEX '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3578359.3593040

1 Introduction

The rise in popularity of trusted execution environments (TEEs) has led to a variety of design proposals [1, 2, 7, 8, 10, 12, 14, 15]. TEEs provide strong confidentiality and integrity guarantees to sensitive user applications (or enclave applications) in a system where the privileged OS is potentially malicious and is not a part of the trusted computing base (TCB). Motivated by the need to support executing unmodified user applications inside the TEE, several TEE frameworks have directed efforts towards the development of trusted runtime systems (TRTS).

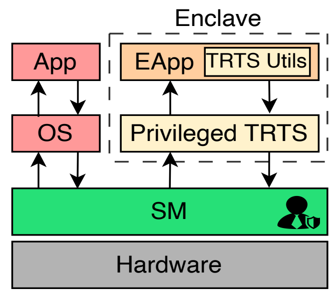


Figure 1. Overview of a generic TEE framework.

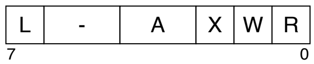
Figure 1 shows a generic view of TEE frameworks [1, 2, 4, 7, 10–13], where the TRTS operates within the enclave to provide familiar OS-like functionality (*e.g.,* virtual memory management, syscalls) to the enclave application and transparently secures interactions between the enclave application and the untrusted OS. TEE frameworks rely on a security monitor (SM) executing at the highest privilege layer (*e.g.,* machine mode in RISC-V [16] or EL3 in ARM [14]) to enforce security guarantees by leveraging existing hardware security primitives [12] or introducing new ones [1]. Interactions between the enclave application and the untrusted host trap into the SM to apply the necessary protection.

While these privileged TRTSs provide necessary functionality to the enclave application, Table 1 shows that they inflate the TCB by tens of thousands of lines of code, and are the cause of demonstrated exploits involving control-flow, memory corruption, and memory leakage attacks. One reason is that, unlike security monitors, TRTSs are typically not designed with formal verification in mind but simply presumed to be non-malicious system software libraries.

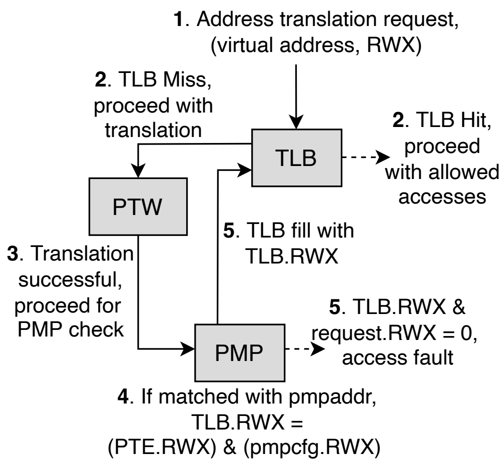
This motivates the need to go beyond a Manichean trust model, where the OS is untrusted and yet the TRTS software is fully trusted. We argue that placing trust in a significantly large piece of software without any precautions increases the chances of the enclave application data being vulnerable in case the TRTS gets compromised.

SysTEX '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0087-3/23/05…$15.00
https://doi.org/10.1145/3578359.3593040

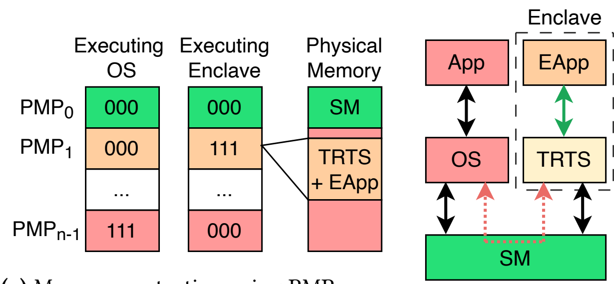


(a) RISC-V `pmpcfg` entry. Dash shows reserved bits.



(b) Address translation process. & indicates logical AND operation. Dashed lines indicate alternative steps.

Figure 2. RISC-V PMP enforcement during the memory address translation process.



(a) Memory protection using PMP. PMP entries show `pmpcfg`.RWX bit values. **(b)** Mode switches.

Figure 3. Privilege mode transitions and memory protection in Keystone.

We propose to enforce the principle of least privilege on TRTS to provide defense-in-depth by protecting enclave application's memory and consequently raising the standards for breaching the TEE's security guarantees. This thwarts Iago attacks that leverage the TRTS's unrestricted access to the enclave application's memory to leak or corrupt it. To achieve this, we propose a lightweight extension to the RISC-V hardware security primitive, physical memory protection (PMP). We illustrate the impact of applying restrictions on a TRTS for the open-source Keystone TEE framework [12].

In the rest of the paper, we provide relevant background on RISC-V and the Keystone framework (§2); make a case for enforcing the least privilege principle on TRTSs and discuss why existing RISC-V hardware security primitives make it inefficient to enforce this (§3). Further, we describe the hardware extensions we introduce to the RISC-V security primitive (§4) and extend the Keystone TEE framework to enforce the least privilege principle, thereby introducing *Keystone-LP* (§5). Lastly, we discuss several classes of attacks, whether Keystone-LP can prevent them, and how our hardware mechanism can extend to ARM TrustZone. (§6)

2 Background

This section provides an overview of RISC-V PMP and the Keystone framework built on top of it.

2.1 RISC-V PMP

RISC-V provides a hardware security primitive, Physical Memory Protection (PMP), to protect physical memory ranges from lower privilege modes. PMP rules are configured by the highest privilege mode in RISC-V *i.e.,* machine mode (`M-mode`), and enforced on all user mode (`U-mode`) and supervisor mode (`S-mode`) memory accesses. Each PMP rule is specified using a pair of registers - the address range requiring protection in a `pmpaddr` register and a set of read, write, and execute (`RWX`) permissions applicable for this memory region in a `pmpcfg` register (see Figure 2a) (which also specifies an addressing mode (A) and the locked bit (L) [1]). PMP uses a priority mechanism; during each PMP check, the first PMP entry matching the address is used to apply permissions.

For each memory access request by the processor, the translation lookaside buffer (TLB) is probed to retrieve the address translation (step 1 in Figure 2b). On a TLB miss, the page table walker (PTW) performs the virtual to physical address translation (step 2). On a successful address translation (step 3), the PMP is checked to validate the memory access for the corresponding physical address (step 4). The PMP computes `TLB.RWX` permissions as the logical AND of the permissions from the page table entry, `PTE.RWX`, with the `pmpcfg.RWX` permissions (step 4). If the requested `RWX` permissions are a subset of `TLB.RWX` permissions, a TLB fill occurs reflecting the `TLB.RWX` permissions (step 5). However, if the requested access violates the PMP rules, an access fault exception is raised [16] (alternative step 5). On a TLB hit, the memory access proceeds without further PMP checks [5] (alternative step 2). It is thus crucial to perform a TLB flush on each PMP rule update.

On a TLB hit, the permissions cached in the TLB are still applicable. In RISC-V, all user pages (`PTE.U` field is set in the page table) are only accessible in `U-mode`; `S-mode` is prohibited from accessing them. However, `S-mode` can set the `SUM` (permit Supervisor User Memory access) bit in the supervisor status (`sstatus.SUM`) register and gain read or write access to all user pages [16]. The aim of this protection is to prevent the OS from unintentionally affecting user pages.

[1] PMP checks can be applicable to `M-mode` using the locked bit, but this case is not relevant for the purpose of this paper.

2.2 Keystone

Keystone [12] leverages PMP to implement enclaves and isolate them from the rest of the system, including the untrusted OS. As shown in Figure 3a, Keystone's SM configures PMP to protect its own physical memory region (PMP0), allows the enclave (enclave application and TRTS) to access the enclave memory region (PMP1) but prevents the enclave from accessing any other physical memory (PMPn). On a context switch from the enclave to the untrusted OS, the SM updates PMP to protect the memory region of the enclave (PMP1) and allows the untrusted OS access to the rest of the memory (PMPn). Thus, Keystone uses one PMP entry per enclave, and two PMP entries for protecting the SM and allowing the OS to access the rest of the memory; it can support $(n-2)$ enclaves simultaneously with an n-entry PMP.

Keystone's TRTS, called Eyrie, performs resource management for the enclave application, and provides an edge call interface to enable interactions between the enclave application and the untrusted OS. The TRTS securely shares only the necessary data through a shared buffer with the untrusted OS (further referred to as *shared buffer with host*).

Figure 3b shows the transitions in which the SM modifies the PMP protections (dotted red transitions) *i.e.,* transitions between trusted enclaves and untrusted host. Whereas, there is no PMP update in the rest of the transitions shown in the figure including transitions between the TRTS and enclave application (transition marked as green).

In the next section, we discuss why having distinct PMP permissions for the TRTS and enclave application is important for the enclave application's security but impractical with the current PMP design.

3 Motivation

This section argues that allowing the TRTS full access to the enclave application's memory violates the principle of least privilege. As it increases the trusted code base (TCB), it potentially introduces bugs that can be exploited to leak enclave application's sensitive data or to corrupt its memory.

3.1 The challenges with TRTS

Existing TEE frameworks offer flexibility in choosing a TRTS, which presents an interesting trade-off between functionality and security. An enclave deployment gets to choose whether to include a full operating system [1], a library OS [2, 4] to provide most OS functionality, or a minimal runtime for securing interactions with the untrusted OS and performing resource management [12]. The larger the TRTS, the less efforts needed to port existing applications to TEE frameworks.

Table 1. Table showing code base sizes of existing TRTS (excluding libOS) and exploited vulnerabilities.

Runtime	kLoC	Discovered vulnerabilities
Keystone [12]	4.6	Iago attack [3]
GrapheneSGX [4]	22.0	Iago attack [3] Integer overflow [3]
Overshadow [7]	13.1	Impersonation attack [18]
Haven LibOS [2]	23.1	Not open-source

Indeed, Table 1 shows that including tens of thousands of lines of code in the TCB increases the likelihood of introducing exploitable bugs and requires careful instrumentation to protect against Iago attacks [1, 4, 6, 7, 12].

The alternative would be to follow the principle of least privilege by restricting the TRTS from accessing application-level sensitive data. This could potentially prevent TRTS vulnerabilities from compromising confidentiality or integrity of the enclave application.

3.2 The limitations of RISC-V PMP

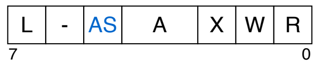
The design of RISC-V PMP aims to protect memory regions from all execution modes except `M-mode` but makes no distinction between `S-mode` and `U-mode` while enforcing PMP rules. The current PMP mechanism could be used to enforce the least privilege principle on the TRTS but this would be inefficient as it would require (1) a trap into `M-mode` for every transition between the TRTS and the enclave application during execution, (2) to reconfigure PMP entries as part of the transition, which would further require (3) a TLB flush. In effect, the transition between enclave application and the TRTS would be as expensive as the transition between the enclave and the host environment.

4 RISC-V Hardware Extensions

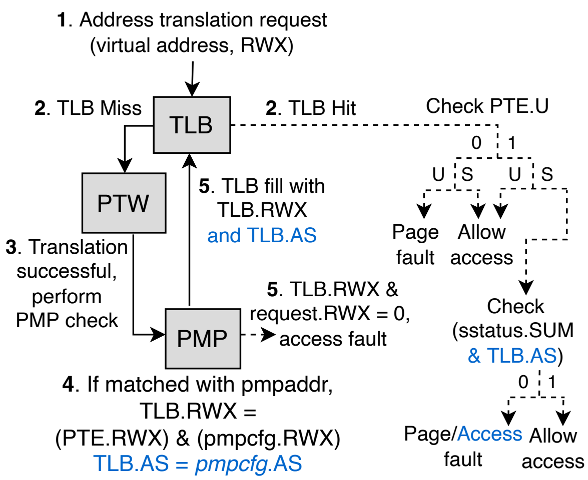
To avoid the PMP limitations discussed in §3.2, we propose to add an additional bit in each `pmpcfg` entry to enforce protection on `U-mode` memory from `S-mode` software, with implications on the TLB design. We use this new feature to enforce the principle of least privilege on the privileged TRTS within the Keystone TEE framework (§5).

Figure 4a illustrates our extension to the `pmpcfg` entries in RISC-V in the form of a new bit *i.e.,* `pmpcfg.AS` (*Allow Supervisor* access) that represents an `M-mode`-governed equivalent of `sstatus.SUM`, which, unlike `sstatus.SUM`, cannot be controlled by system software. The `pmpcfg.RWX` permissions apply as usual on `U-mode` accesses. The new `pmpcfg.AS` bit specifies whether the physical memory access granted by PMP rules extends to `S-mode`. If this bit is clear, `S-mode` software is not allowed to access the corresponding memory region as per the PMP rule. We use one of the reserved bits in the `pmpcfg` entry to implement `pmpcfg.AS`.

In §2, we discussed the role of a PMP check in the address translation process and the `sstatus.SUM` bit in RISC-V, and in particular that PMP checks are bypassed on TLB hits. Whenever `S-mode` enables the `sstatus.SUM` bit, all translations cached in the TLB for user pages (*i.e.,* `PTE.U` is set)



(a) Proposed extension to RISC-V pmpcfg entry *i.e.,* Allow Supervisor bit (`pmpcfg.AS`).



(b) Address translation with proposed extensions (marked in blue). & indicates logical AND operation. Dashed lines indicate alternative steps. (0/1) and (U/S) indicate the sequence of steps taken depending on whether the value is (clear/set) and whether the mode is (`U-mode`/`S-mode`) respectively.

Figure 4. RISC-V PMP enforcement during the memory address translation process with `pmpcfg.AS`.

can be used by `S-mode` to access user memory (potentially enclave application's memory) without going through PMP validation. A straightforward, but architecturally expensive, design to prevent such PMP rule violations could be to maintain separate `U-mode` and `S-mode` TLBs.

Figure 4b describes our proposed TLB extension, which introduces a new *Allow Supervisor* bit in each TLB entry to mirror the same bit from the `pmpcfg` entry. The `TLB.AS` determines whether `S-mode` may access an entry marked with `PTE.U`, based on PMP rules. During a PMP check, the RWX permissions are computed as done in the original PMP design (§2.1). In addition, the `TLB.AS` is set to the same value as `pmpcfg.AS` (step 4). Figure 4b shows the sequence of steps taken on a TLB hit (alternative step 2). On a TLB hit from `S-mode`, the `TLB.AS` is checked in addition to the `sstatus.SUM` bit to determine if the access is legitimate. If the access is not legitimate, a clear `TLB.AS` leads to an access fault. Whereas if the access fails due to a clear `sstatus.SUM` bit, a page fault is raised. The `TLB.AS` is only relevant when the current mode is `S-mode` and the `PTE.U` bit is set. The hardware cost to implement our mechanism is an additional bit per TLB entry (in the entire TLB hierarchy) and an additional comparison on the critical path.

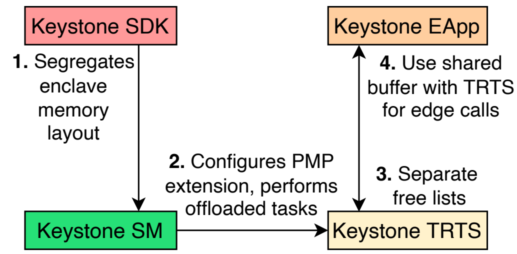


Figure 5. Modifications to Keystone at various steps during enclave setup and execution.

5 Keystone-LP

We have implemented the proposed RISC-V hardware extensions in QEMU and modified the open-source Keystone framework to accommodate the proposed PMP design and enforce the least privilege principle. This requires changes to the TRTS, which previously assumes that it has direct and unrestricted access to the enclave application's memory.

We call the resulting framework *Keystone-LP i.e.,* Keystone with a least privileged TRTS. Keystone-LP works with unmodified Keystone enclave applications enabling them to successfully request services from the TRTS without granting it access to enclave application's memory. Our implementation adds 310 LoC to the original Keystone framework, out of which 234 LoC belongs to the TCB [9].

Note that our goal is *not* to remove the TRTS from the enclave application's TCB, but to enforce the least privilege principle on the TRTS.

We propose three major amendments to Keystone to enable the TRTS to continue providing services to the enclave application without accessing the enclave application's memory directly, thereby enforcing the least privilege principle: (1) segregating the TRTS's and the enclave application's physical memory regions to enforce clear protection rules, (2) offloading some of the TRTS's crucial responsibilities to the SM, and (3) creating a secure communication channel between the TRTS and the enclave application to selectively share data.

Figure 5 shows the chronological order of enclave (enclave application and the TRTS) setup and execution and highlights the changes in Keystone-LP, in particular the SM, the Eyrie RT, and Keystone's SDK.

5.1 Initial enclave setup

During the initial enclave setup, the untrusted Keystone SDK deploys the enclave, *i.e.,* loads TRTS and enclave application binaries into the contiguous physical memory region reserved for the enclave, creates page table mappings, and sets up a free memory region for the enclave. Figure 6 (left) shows the enclave physical memory layout in Keystone. Since an enclave is comprised of both the TRTS and the enclave application, they have a combined free memory region, and the page table is spread out across the entire memory region. The

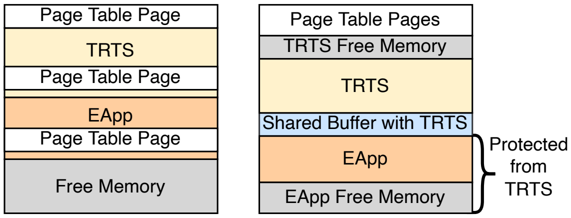


Figure 6. Enclave memory layout in Keystone (left) and Keystone-LP (right).

miscellaneous nature of the enclave memory layout leads to a lack of clear boundaries between the physical memory regions used by the enclave application and the TRTS. Thus, it becomes challenging to isolate the two memory regions using PMP which protects one contiguous memory region through each entry.

Figure 6 shows the physical memory segregation that provides clear boundaries between the enclave application's memory region and the TRTS's memory region, and thus helps enforce least privilege principle in Keystone-LP. Each enclave is therefore bifurcated into two regions: one that is available to the TRTS and the enclave application, and one that is protected from the TRTS.

Once, the enclave setup is complete, the SDK provides all the necessary information regarding the enclave's memory layout to the SM through Keystone-LP's kernel driver.

5.2 Enclave memory protection by the SM

During enclave creation and context switches between the untrusted OS and the enclave, Keystone's SM configures the PMP to protect the enclave's physical memory region.

Keystone-LP's SM uses the PMP extension proposed in §4 to protect enclave application's memory from the TRTS.

Figure 7 shows the `pmpcfg.RWX` bits along with the proposed `pmpcfg.AS` configured by Keystone-LP's SM. As is the case in Keystone, in the highest priority entry, PMP0, the SM protects its memory by clearing `pmpcfg.RWX` and `pmpcfg.AS`. In PMP1, the SM protects the enclave application's memory region from the TRTS, by clearing the `pmpcfg.AS`. In PMP2, the SM configures the rest of the enclave memory region to be accessible by both `U-mode` and `S-mode` by setting `pmpcfg.RWX`and `pmpcfg.AS`. Based on `pmpcfg.AS`, the TRTS which executes in `S-mode` is not allowed to access the enclave application's memory region, but is allowed to access the rest of the enclave's memory. Note that the TRTS's memory is protected from the enclave application through virtual memory, and the shared buffer is the only part of enclave memory accessible by both the TRTS and the enclave application.

Figure 7 also shows the state of PMP when there is a context switch from the enclave to the untrusted OS.

Keystone-LP uses two PMP entries per enclave whereas the original Keystone uses only one entry per enclave. With

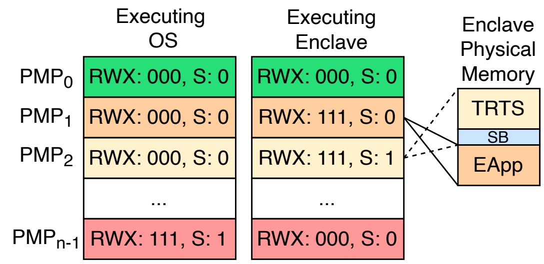


Figure 7. Memory protection using extended PMP. SB = Shared Buffer.

recent RISC-V specifications that state the support for up to 64 PMP entries [16], Keystone-LP should be able to support up to 31 concurrently running enclaves.

5.3 Enclave management by the TRTS

When a TRTS instance boots, it allocates a stack region for the enclave application. Further, throughout the course of enclave execution, the TRTS services page faults and allocates pages as necessary. In both of these instances, Keystone's TRTS accesses the pages being allocated. Firstly, it maintains a free page linked list in the free pages *i.e.,* each free page has a pointer to the next one. Secondly, it zeroes out every free page whenever it gets allocated.

Keystone-LP offloads the zeroing of free memory pages to the SM; this is required as the TRTS no longer has access to the enclave application's memory. The SM clears the eapp's free memory region during enclave creation, so there is no need for the TRTS to repeat it. Further, we implement using an additional data structure in the TRTS to keep track of free pages. We track the free page region in a bit-vector format to denote which pages are free and which are used. The TRTS must track two free page regions - one for itself and one for the enclave application. In theory, the TRTS could use Keystone's free page list mechanism. However, to keep the same code base for managing both free lists, we use the bit-vector approach for the TRTS's memory management too. In addition, the TRTS also keeps track of a dirty list *i.e.,* all the deallocated pages of the enclave application. Keystone-LP's SM provides a service to clear these pages when there are none available in the enclave application's free list.

In Keystone, the TRTS is also responsible for page swapping. This requires access to enclave application's pages. Future work could enable the SM to provide a page-swap API and ensure confidentiality and integrity of the enclave application's pages [7].

5.4 Edge calls by the enclave application

After protecting the enclave application's memory region from the TRTS, any attempt by the TRTS to access enclave application's memory will cause an access fault (Figure 4a). As a consequence, the enclave application's attempts to request services from the TRTS via the edge call interface will raise access fault exceptions when the TRTS's attempts to access the enclave application's memory (*e.g.,* to read the arguments of the edge call).

Keystone-LP implements an TRTS-oblivious mechanism to enable edge calls via a shared buffer between the enclave application and the TRTS (Figure 6) using a mechanism that parallels the *shared buffer with host* already present in the original Keystone. Whenever an enclave application wishes to invoke an edge call, the arguments are first copied to the *shared buffer with TRTS*, and the return value address is set to a location in the *shared buffer with TRTS*. The TRTS copies the arguments of the edge call to the *shared buffer with host* as it would in Keystone. Though this mechanism results in an additional copy of arguments and return values on every edge call, it allows securely exposing data to the TRTS on a need-to-know basis. For performance-sensitive enclave applications that potentially invoke a large volume of edge calls, a simple solution is to offer a choice between security from the TRTS or performance, trading off the other.

6 Discussion

Enforcing the least privilege principle on the TRTS improves confidentiality towards the enclave application and thwarts several attacks where the TRTS is a confused deputy under the control of a malicious OS. Keystone-LP's use of an explicit *shared buffer with TRTS* prevents a naive TRTS from leaking data from the enclave application's memory region.

Our mechanism does not remove the TRTS from the TCB. Several Iago attacks, such as described in [3], would still succeed even with Keystone-LP where a compromised TRTS misleads the enclave application to expose confidential data through the *shared buffer with TRTS*. Further, Keystone-LP doesn't prevent controlled-side channel attacks [17] or control-flow hijacking attacks.

The hardware extension proposed for RISC-V PMP is also applicable to ARM TrustZone [14] which follows a two-world view: all accesses in the memory system are already identified using an `NS-bit` as either secure or non-secure world accesses. An additional bit similar to `pmpcfg.AS` can further help distinguish between various privilege layers inside the secure world to enforce least privilege.

7 Conclusion

In this paper, we proposed a simple extension to the RISC-V hardware security primitive, PMP, to restrict memory accesses of system software. We further used the proposed extension to enforce the least privilege principle on Keystone's trusted runtime, Eyrie, and provided a proof-of-concept implementation to support executing unmodified Keystone enclave applications.

The authors thank the anonymous reviewers for their feedback. This work is supported in part by the Microsoft-EPFL Joint Research Center and a VMware faculty award.

References

[1] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves.. In *Proceedings of the 30th USENIX Security Symposium.* 1073–1090.

[2] Andrew Baumann, Marcus Peinado, and Galen C. Hunt. 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.* 33, 3 (2015), 8:1–8:26.

[3] Jo Van Bulck, David F. Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. 2019. A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes.. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS).* 1741–1758.

[4] Chia che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin John, Harry A. Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald E. Porter. 2014. Cooperation and security isolation of library OSes for multi-process applications.. In *Proceedings of the 2014 EuroSys Conference.* 9:1–9:14.

[5] Kevin Cheang, Cameron Rasmussen, Dayeol Lee, David W. Kohlbrenner, Krste Asanovic, and Sanjit A. Seshia. 2022. Verifying RISC-V Physical Memory Protection. *CoRR* abs/2211.02179 (2022).

[6] Stephen Checkoway and Hovav Shacham. 2013. Iago attacks: why the system call API is a bad untrusted RPC interface.. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XVIII).* 253–264.

[7] Xiaoxin Chen, Tal Garfinkel, E. Christopher Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey S. Dwoskin, and Dan R. K. Ports. 2008. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems.. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XIII).* 2–13.

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016 (2016), 86.

[9] Albert Danial. 2021. *cloc: v1.92.* https://doi.org/10.5281/zenodo.5760077

[10] Google. 2023. *Google Asylo: An open and flexible framework for enclave applications.* https://asylo.dev/

[11] Intel. 2023. *Intel Software Guard Extensions – Get Started with the SDK.* https://software.intel.com/en-us/sgx/sdk

[12] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. 2020. Keystone: an open framework for architecting trusted execution environments.. In *Proceedings of the 2020 EuroSys Conference.* 38:1–38:16.

[13] Microsoft. 2023. *Microsoft Open Enclave SDK.* https://openenclave.io/sdk/

[14] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51, 6 (2019), 130:1–130:36.

[15] AMD SEV-SNP. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* (2020).

[16] Andrew Waterman, Krste Asanović, and John Hauser. 2021. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203.* RISC-V International. https://github.com/riscv/riscv-isa-manual

[17] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems.. In *IEEE Symposium on Security and Privacy.* 640–656.

[18] Siqi Zhao and Xuhua Ding. 2017. On the Effectiveness of Virtualization Based Memory Isolation on Multicore Platforms.. In *Proceedings of the 2017 IEEE European Symposium on Security and Privacy (Euro S&P).* 546–560.